Chapter 6 Error Recovery in CUP

Any parser must be able to cope with syntactically invalid input. It is normally unsatisfactory for the parser to just terminate on detecting an error. Errors must be recovered from in some way, producing an error message, and continuing the process of parsing until the end of input is reached.

CUP has an intrinsic “error” symbol (neither exactly a terminal nor a nonterminal). When an error is detected, a portion of the top of stack, and a portion of the following input are deleted, and replaced by the “error” symbol. Thus the “error” symbol effectively matches arbitrary input surrounding the position at which the error was detected. The portion of stack deleted of course corresponds to input already parsed.

The error symbol can be used on the right hand side of grammar rules, as in

```
Stmt::= error NEWLINE ;
```

When the parser cannot perform a shift, reduction, or accept, then the parser enters error mode, the parser generates an error message, cuts the parser stack back until it has a state that can shift the symbol “error”, and shifts error onto the stack. (If there is no state that can shift error, then the parser aborts.).

It then deletes tokens until it can successfully parse error_sync_size() tokens without generating another error. The default version of the method error_sync_size() returns the value 3, so that normally the parser has to be able to consume 3 tokens before the error is considered to be “recovered from”. To alter the number of tokens that need to be parsed to recover from an error, you can override the error_sync_size() method.

Once the parser is placed into a configuration that has an immediate error recovery (by popping the stack to the first such state), the parser begins deleting tokens to find a point at which the parse can be continued. After discarding each token, the parser attempts to parse ahead in the input (without executing any embedded semantic actions). If the parser can successfully parse past the required number of tokens, then the input is backed up to the point of recovery and the parse is resumed normally (executing all actions). If the parse cannot be continued far enough, then another token is discarded and the parser again tries to parse ahead. If the end of input is reached without making a successful recovery (or there was no suitable error recovery state found on the parse stack to begin with) then error recovery fails.

The sort of error recovery available in CUP, allows for little more than what is called the "panic mode" of error recovery, where input is consumed until the parser reaches a significant token, such as the end of line, a token that can terminate or follow a statement, or a token that can start a new statement.

In many cases, there are tokens that are known to be clear markers for the end of a statement. For example

```
Stmt::= error NEWLINE;
```

will consume input until we have a newline (assuming newlines are syntactically important).

```
Stmt::= error SEMICOLON;
```

```
Stmt::= error RIGHTCURLY;
```

will consume input until we have a “;” or “}”, both of which are clear markers for the ends of statements in Java and C.

Sometimes the marker is not part of the statement itself, but is a token used to separate statements. For example

```
Stmt::= error;
```

will consume input until we have a token that can follow a statement. This is useful in Pascal, where the ";" separates statements, rather than terminating them.

In some languages all statements have clear markers to start the statement. For example

```
Stmt::= error Stmt;
```

will consume text until it finds a token that can start a new statement.

Example

A bottom up parse of the invalid input

```
- a * b + c * - d \n
```

with the grammar used as an example of bottom up parsing in chapter 2 successfully parses until it generates

$0 SL1 E5 +23 T25 *17 - Error (Cannot parse due to invalid input).

At this stage it cuts the stack back to

$0 SL1 -

then pushes "error" onto the stack

$0 SL1 error 12 -

then deletes input until it gets a token it can shift onto the stack, namely \n.

$0 SL1 error 12 \n

It shifts the \n onto the stack

$0 SL1 error 12 \n 13 $

Then reduces by the rule Program → Program error \n.

$0 SL1 $

It has now "recovered" from the error.

Example

It is very important that the error symbol be used very sparingly in your grammar. The manner in which the parser cuts back the stack to the first place at which error can be shifted, fixes the parser into trying to correct the error for a specific construct, and this may not be appropriate. Only use error recovery for major constructs such as statements, and only scan ahead for very well defined markers for the end of the statement.

Consider the grammar

```
terminal LEFTBRACE, RIGHTBRACE, ASSIGN, SEMICOLON, COMMA;
terminal String IDENT;

nonterminal ProgramNode Program;
nonterminal DeclListNode DeclList;
nonterminal DeclNode Decl;
nonterminal TypeNode Type;
nonterminal DeclrListNode DeclrList;
nonterminal DeclrNode Declr;
nonterminal ExprNode Expr;
nonterminal ExprListNode ExprList;

start with Program;
```

```
Program::=
          DeclList:declList
          {:
          RESULT = new ProgramNode( declList );
          :}
     ;

DeclList::=
          {:
          RESULT = new DeclListNode();
          :}
     |
          DeclList:declList Decl:decl
          {:
          declList.addElement( decl );
          RESULT = declList;
          :}
     ;

Decl::=
          Type:type DeclrList:declrList SEMICOLON
          {:
          RESULT = new VariableDeclNode( type, declrList );
          :}
     |
          error SEMICOLON
          {:
          RESULT = new ErrorDeclNode( "DeclError...;" );
          :}
     ;

Type::=
          IDENT:ident
          {:
          RESULT = new TypeIdentNode( ident );
          :}
     ;

DeclrList::=
          Declr:declr
          {:
          RESULT = new DeclrListNode( declr );
          :}
     |
          DeclrList:declrList COMMA Declr:declr
          {:
          declrList.addElement( declr );
          RESULT = declrList;
          :}
     ;

Declr::=
          IDENT:ident ASSIGN Expr:expr
          {:
          RESULT = new InitDeclrNode( ident, expr );
          :}
     |
          IDENT:ident
          {:
          RESULT = new UninitDeclrNode( ident );
```

```
            :}
      |
            error
            {:
            RESULT = new ErrorDeclrNode( "DeclrError..." );
            :}
      ;

Expr::=
            LEFTBRACE ExprList:exprList RIGHTBRACE
            {:
            RESULT = new CompoundExprNode( exprList );
            :}
      |
            LEFTBRACE error RIGHTBRACE
            {:
            RESULT = new ErrorExprNode( "{ ExprListError ... }" );
            :}
      |
            IDENT:ident
            {:
            RESULT = new VariableExprNode( ident );
            :}
      ;

ExprList::=
            Expr:expr
            {:
            RESULT = new ExprListNode( expr, null );
            :}
      |
            Expr:expr COMMA ExprList:exprList
            {:
            RESULT = new ExprListNode( expr, exprList );
            :}
      ;
```

In fact this grammar has too many grammar rules involving “error”.

The rule

```
Expr::=
            LEFTBRACE error RIGHTBRACE
            {:
            RESULT = new ErrorExprNode( "{ ExprListError ... }" );
            :}
```

means that if the parser finds a “{” in an expression, then detects a syntax error, the parser will scan ahead trying to match a “}”. Perhaps the syntax error was caused by missing out the “}”. Worse still, perhaps there is no “}” in the following input. The parser will eat up the remaining input, then generate an error it cannot recover from.

The rule

```
Declr::=
            error
            {:
            RESULT = new ErrorDeclrNode( "DeclrError..." );
            :}
      ;
```

has similar failings, but is not quite as harmful. A missing ";" will mean that the parser scans ahead for a "," or ";" (since that is what can follow a "Declr"). It might eat up the start of the next construct, and process the remainder of that construct as part of the current construct.

What happens for input such as

```
int a b = { c, d, e }, f, g
int h, i;
int j = { k;
int l = { m, n }, o;
```

It produces error messages

```
Programs/invalid/program.in ( 1 ): Syntax Error
int a b = { c, d, e }, f, g
      ^
Programs/invalid/program.in ( 1 ): Syntax Error
int a b = { c, d, e }, f, g|int h, i;|int
                      ^      |         |
Programs/invalid/program.in ( 2 ): Syntax Error
int a b = { c, d, e }, f, g|int h, i;|int j = { k;|
                           |^^^      |           |
Programs/invalid/program.in ( 3 ): Syntax Error
int a b = { c, d, e }, f, g|int h, i;|int j = { k;|int l = { m, n }, o
                           |         |           ^|
```

It ends up parsing the input as

```
int DeclrError..., d, DeclrError..., f, DeclrError..., i;
int j = { ExprListError ... }, o;
```

We get the following parse. I have made error_sync_size() return 1, to avoid double parsing of input.

$ 0							
$ 0	DeclL 2						
$ 0	DeclL 2	ID 5					
$ 0	DeclL 2	Type 3					
$ 0	DeclL 2	Type 3	ID 9				
$ 0	DeclL 2	Type 3					
$ 0	DeclL 2	Type 3	error 8				
$ 0	DeclL 2	Type 3	error 8				
$ 0	DeclL 2	Type 3	error 8				
$ 0	DeclL 2	Type 3	error 8				
$ 0	DeclL 2	Type 3	error 8				
$ 0	DeclL 2	Type 3	Dclr 10				
$ 0	DeclL 2	Type 3	DclrL 11				
$ 0	DeclL 2	Type 3	DclrL 11	, 12			
$ 0	DeclL 2	Type 3	DclrL 11	, 12	ID 9		
$ 0	DeclL 2	Type 3	DclrL 11	, 12	Dclr 14		
$ 0	DeclL 2	Type 3	DclrL 11				
$ 0	DeclL 2	Type 3	DclrL 11	, 12			
$ 0	DeclL 2	Type 3	DclrL 11	, 12	ID 9		
$ 0	DeclL 2	Type 3	DclrL 11	, 12			
$ 0	DeclL 2	Type 3	DclrL 11	, 12	error 8		
$ 0	DeclL 2	Type 3	DclrL 11	, 12	error 8		
$ 0	DeclL 2	Type 3	DclrL 11	, 12	Dclr 14		
$ 0	DeclL 2	Type 3	DclrL 11				
$ 0	DeclL 2	Type 3	DclrL 11	, 12			
$ 0	DeclL 2	Type 3	DclrL 11	, 12	ID 9		
$ 0	DeclL 2	Type 3	DclrL 11	, 12	Dclr 14		
$ 0	DeclL 2	Type 3	DclrL 11				

ID int	Reduce	DeclL ::=
	Shift	ID 5
ID a	Reduce	Type ::= ID
	Shift	ID 9
ID b	Error	Pop Stack
	Shift	error 8
	Error	Consume ID b
=	Error	Consume =
{	Error	Consume {
ID c	Error	Consume ID c
,	Reduce	Dclr ::= error
	Reduce	DclrL ::= Dclr
	Shift	, 12
ID d	Shift	ID 9
,	Reduce	Dclr ::= ID
	Reduce	DclrL ::= DclrL , Dclr
	Shift	, 12
ID e	Shift	ID 9
}	Error	Pop Stack
	Shift	error 8
	Error	Consume }
,	Reduce	Dclr ::= error
	Reduce	DclrL ::= DclrL , Dclr
	Shift	, 12
ID f	Shift	ID 9
,	Reduce	Dclr ::= ID
	Reduce	DclrL ::= DclrL , Dclr
	Shift	, 12

$ 0	DeclL 2	Type 3	DclrL 11	, 12				ID g	Shift	ID 9
$ 0	DeclL 2	Type 3	DclrL 11	, 12	ID 9			ID int	Error	Pop Stack
$ 0	DeclL 2	Type 3	DclrL 11	, 12					Shift	error 8
$ 0	DeclL 2	Type 3	DclrL 11	, 12	error 8				Error	Consume ID int
$ 0	DeclL 2	Type 3	DclrL 11	, 12	error 8			ID h	Error	Consume ID h
$ 0	DeclL 2	Type 3	DclrL 11	, 12	error 8			,	Reduce	Dclr ::= error
$ 0	DeclL 2	Type 3	DclrL 11	, 12	Dclr 14				Reduce	DclrL ::= DclrL , Dclr
$ 0	DeclL 2	Type 3	DclrL 11						Shift	, 12
$ 0	DeclL 2	Type 3	DclrL 11	, 12				ID i	Shift	ID 9
$ 0	DeclL 2	Type 3	DclrL 11	, 12	ID 9			;	Reduce	Dclr ::= ID
$ 0	DeclL 2	Type 3	DclrL 11	, 12	Dclr 14				Reduce	DclrL ::= DclrL , Dclr
$ 0	DeclL 2	Type 3	DclrL 11						Shift	; 13
$ 0	DeclL 2	Type 3	DclrL 11	; 13				ID int	Reduce	Decl ::= Type DclrL ;
$ 0	DeclL 2	Decl 6							Reduce	DeclL ::= DeclL Decl
$ 0	DeclL 2								Shift	ID 5
$ 0	DeclL 2	ID 5						ID j	Reduce	Type ::= ID
$ 0	DeclL 2	Type 3							Shift	ID 9
$ 0	DeclL 2	Type 3	ID 9					=	Shift	= 15
$ 0	DeclL 2	Type 3	ID 9	= 15				{	Shift	{ 16
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16			ID k	Shift	ID 18
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	ID 18		;	Reduce	Exp ::= ID*
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	Exp 21			Error	Pop Stack
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16				Shift	error 20
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	error 20			Error	Consume ;
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	error 20		ID int	Error	Consume ID int
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	error 20		ID 1	Error	Consume ID 1
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	error 20		=	Error	Consume =
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	error 20		{	Error	Consume {
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	error 20		ID m	Error	Consume ID m
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	error 20		,	Error	Consume ,
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	error 20		ID n	Error	Consume ID n
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	error 20		}	Shift	} 24
$ 0	DeclL 2	Type 3	ID 9	= 15	{ 16	error 20	} 24	,	Reduce	Exp ::= { error }
$ 0	DeclL 2	Type 3	ID 9	= 15	Exp 17				Reduce	Dclr ::= ID = Exp
$ 0	DeclL 2	Type 3	Dclr 10						Reduce	DclrL ::= Dclr
$ 0	DeclL 2	Type 3	DclrL 11						Shift	, 12
$ 0	DeclL 2	Type 3	DclrL 11	, 12				ID o	Shift	ID 9
$ 0	DeclL 2	Type 3	DclrL 11	, 12	ID 9			;	Reduce	Dclr ::= ID
$ 0	DeclL 2	Type 3	DclrL 11	, 12	Dclr 14				Reduce	DclrL ::= DclrL , Dclr
$ 0	DeclL 2	Type 3	DclrL 11						Shift	; 13
$ 0	DeclL 2	Type 3	DclrL 11	; 13				$	Reduce	Decl ::= Type DclrL ;
$ 0	DeclL 2	DecL 6							Reduce	DeclL ::= DeclL Decl
$ 0	DeclL 2								Reduce	Prog ::= DeclL
$ 0	Prog 1								Shift	$ 26
$ 0	Prog 1	$ 26							Reduce	$Start ::= Prog $
$ 0	$Start-1								Accept	

You can find out exactly what it does by running the parser in debug mode. The method error_sync_size() has been set to return 1.

```
# Initializing parser
Obtain token IDENT "int"
# Current token is IDENT
# Reduce by rule DeclList ::=
# Shift nonterminal DeclList to push state #2
# Shift token IDENT to push state #5
Obtain token IDENT "a"
```

```
# Current token is IDENT
# Reduce by rule Type ::= IDENT
# Shift nonterminal Type to push state #3
# Shift token IDENT to push state #9
Obtain token IDENT "b"
# Current token is IDENT
# Enter error recovery
============ Parse Stack Dump ============
   Terminal: EOF   State: 0
Nonterminal: DeclList   State: 2
Nonterminal: Type  State: 3
   Terminal: IDENT State: 9
==========================================
#     Enter Find recovery state on stack
#     Find recovery config: Pop stack by one, state was # 9
#     Exit Find recovery config: Recover state found (#3)
#     Shift error to push state #8
# Error recovery: read lookahead
Obtain token ASSIGN "="
# Error recovery: Trying to parse ahead
# Error recovery: Consume token IDENT
Obtain token LEFTBRACE "{"
# Error recovery: Trying to parse ahead
# Error recovery: Consume token ASSIGN
Obtain token IDENT "c"
# Error recovery: Trying to parse ahead
# Error recovery: Consume token LEFTBRACE
Obtain token COMMA ","
# Error recovery: Trying to parse ahead
# Error recovery: Consume token IDENT
Obtain token IDENT "d"
# Error recovery: Trying to parse ahead
#     Try parse ahead: Reduce by rule Declr ::= error
#     Try parse ahead: Shift nonterminal Declr to push state #10
#     Try parse ahead: Reduce by rule DeclrList ::= Declr
#     Try parse ahead: Shift nonterminal DeclrList to push state #11
#     Try parse ahead shifts token COMMA to push state #12
# Error recovery: Parse-ahead ok, going back to normal parse
#     Parse lookahead: Reparsing saved input with actions
#     Parse lookahead: Current token is COMMA
#     Parse lookahead: Current state is #8
# Reduce by rule Declr ::= error
#     Parse lookahead: Shift nonterminal Declr to push state #10
# Reduce by rule DeclrList ::= Declr
#     Parse lookahead: Shift nonterminal DeclrList to push state #11
# Shift token COMMA to push state #12
#     Parse lookahead: Completed reparse
# Exit error recovery - success
============ Parse Stack Dump ============
   Terminal: EOF   State: 0
Nonterminal: DeclList   State: 2
Nonterminal: Type  State: 3
Nonterminal: DeclrList  State: 11
   Terminal: COMMA State: 12
==========================================
# Shift token IDENT to push state #9
Obtain token COMMA ","
# Current token is COMMA
# Reduce by rule Declr ::= IDENT
# Shift nonterminal Declr to push state #14
# Reduce by rule DeclrList ::= DeclrList COMMA Declr
```

```
# Shift nonterminal DeclrList to push state #11
# Shift token COMMA to push state #12
Obtain token IDENT "e"
# Current token is IDENT
# Shift token IDENT to push state #9
Obtain token RIGHTBRACE "}"
# Current token is RIGHTBRACE
# Enter error recovery
============ Parse Stack Dump ============
   Terminal: EOF   State: 0
Nonterminal: DeclList   State: 2
Nonterminal: Type  State: 3
Nonterminal: DeclrList  State: 11
   Terminal: COMMA State: 12
   Terminal: IDENT State: 9
=========================================
#     Enter Find recovery state on stack
#     Find recovery config: Pop stack by one, state was # 9
#     Exit Find recovery config: Recover state found (#12)
#     Shift error to push state #8
# Error recovery: read lookahead
Obtain token COMMA ","
# Error recovery: Trying to parse ahead
# Error recovery: Consume token RIGHTBRACE
Obtain token IDENT "f"
# Error recovery: Trying to parse ahead
#     Try parse ahead: Reduce by rule Declr ::= error
#     Try parse ahead: Shift nonterminal Declr to push state #14
#     Try parse ahead: Reduce by rule DeclrList ::= DeclrList COMMA Declr
#     Try parse ahead: Shift nonterminal DeclrList to push state #11
#     Try parse ahead shifts token COMMA to push state #12
# Error recovery: Parse-ahead ok, going back to normal parse
#     Parse lookahead: Reparsing saved input with actions
#     Parse lookahead: Current token is COMMA
#     Parse lookahead: Current state is #8
# Reduce by rule Declr ::= error
#     Parse lookahead: Shift nonterminal Declr to push state #14
# Reduce by rule DeclrList ::= DeclrList COMMA Declr
#     Parse lookahead: Shift nonterminal DeclrList to push state #11
# Shift token COMMA to push state #12
#     Parse lookahead: Completed reparse
# Exit error recovery - success
============ Parse Stack Dump ============
   Terminal: EOF   State: 0
Nonterminal: DeclList   State: 2
Nonterminal: Type  State: 3
Nonterminal: DeclrList  State: 11
   Terminal: COMMA State: 12
=========================================
# Shift token IDENT to push state #9
Obtain token COMMA ","
# Current token is COMMA
# Reduce by rule Declr ::= IDENT
# Shift nonterminal Declr to push state #14
# Reduce by rule DeclrList ::= DeclrList COMMA Declr
# Shift nonterminal DeclrList to push state #11
# Shift token COMMA to push state #12
Obtain token IDENT "g"
# Current token is IDENT
# Shift token IDENT to push state #9
Obtain token IDENT "int"
```

```
# Current token is IDENT
# Enter error recovery
============ Parse Stack Dump ============
   Terminal: EOF   State: 0
Nonterminal: DeclList   State: 2
Nonterminal: Type  State: 3
Nonterminal: DeclrList  State: 11
   Terminal: COMMA State: 12
   Terminal: IDENT State: 9
=========================================
#     Enter Find recovery state on stack
#     Find recovery config: Pop stack by one, state was # 9
#     Exit Find recovery config: Recover state found (#12)
#     Shift error to push state #8
# Error recovery: read lookahead
Obtain token IDENT "h"
# Error recovery: Trying to parse ahead
# Error recovery: Consume token IDENT
Obtain token COMMA ","
# Error recovery: Trying to parse ahead
# Error recovery: Consume token IDENT
Obtain token IDENT "i"
# Error recovery: Trying to parse ahead
#     Try parse ahead: Reduce by rule Declr ::= error
#     Try parse ahead: Shift nonterminal Declr to push state #14
#     Try parse ahead: Reduce by rule DeclrList ::= DeclrList COMMA Declr
#     Try parse ahead: Shift nonterminal DeclrList to push state #11
#     Try parse ahead shifts token COMMA to push state #12
# Error recovery: Parse-ahead ok, going back to normal parse
#     Parse lookahead: Reparsing saved input with actions
#     Parse lookahead: Current token is COMMA
#     Parse lookahead: Current state is #8
# Reduce by rule Declr ::= error
#     Parse lookahead: Shift nonterminal Declr to push state #14
# Reduce by rule DeclrList ::= DeclrList COMMA Declr
#     Parse lookahead: Shift nonterminal DeclrList to push state #11
# Shift token COMMA to push state #12
#     Parse lookahead: Completed reparse
# Exit error recovery - success
============ Parse Stack Dump ============
   Terminal: EOF   State: 0
Nonterminal: DeclList   State: 2
Nonterminal: Type  State: 3
Nonterminal: DeclrList  State: 11
   Terminal: COMMA State: 12
=========================================
# Shift token IDENT to push state #9
Obtain token SEMICOLON ";"
# Current token is SEMICOLON
# Reduce by rule Declr ::= IDENT
# Shift nonterminal Declr to push state #14
# Reduce by rule DeclrList ::= DeclrList COMMA Declr
# Shift nonterminal DeclrList to push state #11
# Shift token SEMICOLON to push state #13
Obtain token IDENT "int"
# Current token is IDENT
# Reduce by rule Decl ::= Type DeclrList SEMICOLON
# Shift nonterminal Decl to push state #6
# Reduce by rule DeclList ::= DeclList Decl
# Shift nonterminal DeclList to push state #2
# Shift token IDENT to push state #5
```

```
Obtain token IDENT "j"
# Current token is IDENT
# Reduce by rule Type ::= IDENT
# Shift nonterminal Type to push state #3
# Shift token IDENT to push state #9
Obtain token ASSIGN "="
# Current token is ASSIGN
# Shift token ASSIGN to push state #15
Obtain token LEFTBRACE "{"
# Current token is LEFTBRACE
# Shift token LEFTBRACE to push state #16
Obtain token IDENT "k"
# Current token is IDENT
# Shift token IDENT to push state #18
Obtain token SEMICOLON ";"
# Current token is SEMICOLON
# Reduce by rule Expr ::= IDENT
# Shift nonterminal Expr to push state #21
# Enter error recovery
============ Parse Stack Dump ============
   Terminal: EOF   State: 0
Nonterminal: DeclList   State: 2
Nonterminal: Type  State: 3
   Terminal: IDENT State: 9
   Terminal: ASSIGN     State: 15
   Terminal: LEFTBRACE  State: 16
Nonterminal: Expr  State: 21
=========================================
#     Enter Find recovery state on stack
#     Find recovery config: Pop stack by one, state was # 21
#     Exit Find recovery config: Recover state found (#16)
#     Shift error to push state #20
# Error recovery: read lookahead
Obtain token IDENT "int"
# Error recovery: Trying to parse ahead
# Error recovery: Consume token SEMICOLON
Obtain token IDENT "l"
# Error recovery: Trying to parse ahead
# Error recovery: Consume token IDENT
Obtain token ASSIGN "="
# Error recovery: Trying to parse ahead
# Error recovery: Consume token IDENT
Obtain token LEFTBRACE "{"
# Error recovery: Trying to parse ahead
# Error recovery: Consume token ASSIGN
Obtain token IDENT "m"
# Error recovery: Trying to parse ahead
# Error recovery: Consume token LEFTBRACE
Obtain token COMMA ","
# Error recovery: Trying to parse ahead
# Error recovery: Consume token IDENT
Obtain token IDENT "n"
# Error recovery: Trying to parse ahead
# Error recovery: Consume token COMMA
Obtain token RIGHTBRACE "}"
# Error recovery: Trying to parse ahead
# Error recovery: Consume token IDENT
Obtain token COMMA ","
# Error recovery: Trying to parse ahead
#     Try parse ahead shifts token RIGHTBRACE to push state #24
# Error recovery: Parse-ahead ok, going back to normal parse
```

```
#     Parse lookahead: Reparsing saved input with actions
#     Parse lookahead: Current token is RIGHTBRACE
#     Parse lookahead: Current state is #20
# Shift token RIGHTBRACE to push state #24
#     Parse lookahead: Completed reparse
# Exit error recovery - success
============ Parse Stack Dump ============
   Terminal: EOF   State: 0
Nonterminal: DeclList   State: 2
Nonterminal: Type  State: 3
   Terminal: IDENT State: 9
   Terminal: ASSIGN     State: 15
   Terminal: LEFTBRACE  State: 16
   Terminal: error State: 20
   Terminal: RIGHTBRACE State: 24
=========================================
# Reduce by rule Expr ::= LEFTBRACE error RIGHTBRACE
# Shift nonterminal Expr to push state #17
# Reduce by rule Declr ::= IDENT ASSIGN Expr
# Shift nonterminal Declr to push state #10
# Reduce by rule DeclrList ::= Declr
# Shift nonterminal DeclrList to push state #11
# Shift token COMMA to push state #12
Obtain token IDENT "o"
# Current token is IDENT
# Shift token IDENT to push state #9
Obtain token SEMICOLON ";"
# Current token is SEMICOLON
# Reduce by rule Declr ::= IDENT
# Shift nonterminal Declr to push state #14
# Reduce by rule DeclrList ::= DeclrList COMMA Declr
# Shift nonterminal DeclrList to push state #11
# Shift token SEMICOLON to push state #13
Obtain token EOF ""
# Current token is EOF
# Reduce by rule Decl ::= Type DeclrList SEMICOLON
# Shift nonterminal Decl to push state #6
# Reduce by rule DeclList ::= DeclList Decl
# Shift nonterminal DeclList to push state #2
# Reduce by rule Program ::= DeclList
# Shift nonterminal Program to push state #1
# Shift token EOF to push state #26
Obtain token EOF ""
# Current token is EOF
# Reduce by rule $START ::= Program EOF
# Shift nonterminal $START to push state #-1
```

The rules and action and goto table are as follows.

Rules

```
[0] $START ::= Program EOF
[1] Program ::= DeclList
[2] DeclList ::=
[3] DeclList ::= DeclList Decl
[4] Decl ::= Type DeclrList SEMICOLON
[5] Decl ::= error SEMICOLON
[6] Type ::= IDENT
[7] DeclrList ::= Declr
[8] DeclrList ::= DeclrList COMMA Declr
[9] Declr ::= IDENT ASSIGN Expr
[10] Declr ::= IDENT
[11] Declr ::= error
```

```
[12] Expr ::= LEFTBRACE ExprList RIGHTBRACE
[13] Expr ::= LEFTBRACE error RIGHTBRACE
[14] Expr ::= IDENT
[15] ExprList ::= Expr
[16] ExprList ::= Expr COMMA ExprList
```

Action Table

```
From state #0
     EOF:REDUCE(rule 2) error:REDUCE(rule 2) IDENT:REDUCE(rule 2)
From state #1
     EOF:SHIFT(state 26)
From state #2
     EOF:REDUCE(rule 1) error:SHIFT(state 4) IDENT:SHIFT(state 5)
From state #3
     error:SHIFT(state 8) IDENT:SHIFT(state 9)
From state #4
     SEMICOLON:SHIFT(state 7)
From state #5
     error:REDUCE(rule 6) IDENT:REDUCE(rule 6)
From state #6
     EOF:REDUCE(rule 3) error:REDUCE(rule 3) IDENT:REDUCE(rule 3)
From state #7
     EOF:REDUCE(rule 5) error:REDUCE(rule 5) IDENT:REDUCE(rule 5)
From state #8
     SEMICOLON:REDUCE(rule 11) COMMA:REDUCE(rule 11)
From state #9
     ASSIGN:SHIFT(state 15) SEMICOLON:REDUCE(rule 10) COMMA:REDUCE(rule 10)
From state #10
     SEMICOLON:REDUCE(rule 7) COMMA:REDUCE(rule 7)
From state #11
     SEMICOLON:SHIFT(state 13) COMMA:SHIFT(state 12)
From state #12
     error:SHIFT(state 8) IDENT:SHIFT(state 9)
From state #13
     EOF:REDUCE(rule 4) error:REDUCE(rule 4) IDENT:REDUCE(rule 4)
From state #14
     SEMICOLON:REDUCE(rule 8) COMMA:REDUCE(rule 8)
From state #15
     LEFTBRACE:SHIFT(state 16) IDENT:SHIFT(state 18)
From state #16
     error:SHIFT(state 20) LEFTBRACE:SHIFT(state 16) IDENT:SHIFT(state 18)
From state #17
     SEMICOLON:REDUCE(rule 9) COMMA:REDUCE(rule 9)
From state #18
     RIGHTBRACE:REDUCE(rule 14) SEMICOLON:REDUCE(rule 14) COMMA:REDUCE(rule
14)
From state #19
     RIGHTBRACE:SHIFT(state 25)
From state #20
     RIGHTBRACE:SHIFT(state 24)
From state #21
     RIGHTBRACE:REDUCE(rule 15) COMMA:SHIFT(state 22)
From state #22
     LEFTBRACE:SHIFT(state 16) IDENT:SHIFT(state 18)
From state #23
     RIGHTBRACE:REDUCE(rule 16)
From state #24
     RIGHTBRACE:REDUCE(rule 13) SEMICOLON:REDUCE(rule 13) COMMA:REDUCE(rule
13)
From state #25
     RIGHTBRACE:REDUCE(rule 12) SEMICOLON:REDUCE(rule 12) COMMA:REDUCE(rule
12)
```

```
From state #26
     EOF:REDUCE(rule 0)
```

Goto (Reduce) Table

```
From state #0:
     Program:GOTO(1)
     DeclList:GOTO(2)
From state #1:
From state #2:
     Decl:GOTO(6)
     Type:GOTO(3)
From state #3:
     DeclrList:GOTO(11)
     Declr:GOTO(10)
From state #4:
From state #5:
From state #6:
From state #7:
From state #8:
From state #9:
From state #10:
From state #11:
From state #12:
     Declr:GOTO(14)
From state #13:
From state #14:
From state #15:
     Expr:GOTO(17)
From state #16:
     Expr:GOTO(21)
     ExprList:GOTO(19)
From state #17:
From state #18:
From state #19:
From state #20:
From state #21:
From state #22:
     Expr:GOTO(21)
     ExprList:GOTO(23)
From state #23:
From state #24:
From state #25:
From state #26:
-----------------------------
```

Exercise

Delete the rule

```
Expr::=
          LEFTBRACE error RIGHTBRACE
          {:
          RESULT = new ErrorExprNode( "{ ExprListError ... }" );
          :}
```

And add the rule

```
ExprList::=
          error
          {:
          RESULT = new ErrorExprListNode( "ExprListError ..." );
          :}
```

to the grammar in the previous exercise, and compare parsing for

```
int a = { b, c, d e, f, g };
```